*algorithms*

MDPI

*Article*

# Efficient Data Structures for Range Shortest Unique Substring Queries †

**Paniz Abedin [1], Arnab Ganguly [2], Solon P. Pissis [3,4,*] and Sharma V. Thankachan [1]**

[1] Department of Computer Science, University of Central Florida, Orlando, FL 32816, USA; paniz@knights.ucf.edu (P.A.); sharma.thankachan@ucf.edu (S.V.T.)
[2] Department of Computer Science, University of Wisconsin - Whitewater, Whitewater, WI 53190, USA; gangulya@uww.edu
[3] Life Sciences and Health, CWI, 1098 XG Amsterdam, The Netherlands
[4] Center for Integrative Bioinformatics, Vrije Universiteit, 1081 HV Amsterdam, The Netherlands
* Correspondence: solon.pissis@cwi.nl
† An early version of this paper appeared in the Proceedings of SPIRE 2019.

Received: 9 September 2020; Accepted: 29 October 2020; Published: 30 October 2020


**Abstract:** Let $\mathsf{T}[1,n]$ be a string of length $n$ and $\mathsf{T}[i,j]$ be the substring of $\mathsf{T}$ starting at position $i$ and ending at position $j$. A substring $\mathsf{T}[i,j]$ of $\mathsf{T}$ is a repeat if it occurs more than once in $\mathsf{T}$; otherwise, it is a unique substring of $\mathsf{T}$. Repeats and unique substrings are of great interest in computational biology and information retrieval. Given string $\mathsf{T}$ as input, the Shortest Unique Substring problem is to find a shortest substring of $\mathsf{T}$ that does not occur elsewhere in $\mathsf{T}$. In this paper, we introduce the range variant of this problem, which we call the Range Shortest Unique Substring problem. The task is to construct a data structure over $\mathsf{T}$ answering the following type of online queries efficiently. Given a range $[\alpha,\beta]$, return a shortest substring $\mathsf{T}[i,j]$ of $\mathsf{T}$ with exactly one occurrence in $[\alpha,\beta]$. We present an $\mathcal{O}(n\log n)$-word data structure with $\mathcal{O}(\log_w n)$ query time, where $w=\Omega(\log n)$ is the word size. Our construction is based on a non-trivial reduction allowing for us to apply a recently introduced optimal geometric data structure [Chan et al., ICALP 2018]. Additionally, we present an $\mathcal{O}(n)$-word data structure with $\mathcal{O}(\sqrt{n}\log^{\epsilon} n)$ query time, where $\epsilon>0$ is an arbitrarily small constant. The latter data structure relies heavily on another geometric data structure [Nekrich and Navarro, SWAT 2012].

**Keywords:** shortest unique substring; suffix tree; heavy-light decomposition; range queries; geometric data structures

## 1. Introduction

Finding regularities in strings is one of the main topics of combinatorial pattern matching and its applications [1]. Among the most well-studied types of string regularities is the notion of repeat. Let $\mathsf{T}[1,n]$ be a string of length $n$. A substring $\mathsf{T}[i,j]$ of $\mathsf{T}$ is called a repeat if it occurs more than once in $\mathsf{T}$. The notion of unique substring is dual: it is a substring $\mathsf{T}[i,j]$ of $\mathsf{T}$ that does not occur more than once in $\mathsf{T}$. Computing repeats and unique substrings has applications in computational biology [2,3] and information retrieval [4,5].

In this paper, we are interested in the notion of shortest unique substring. All of the shortest unique substrings of string $\mathsf{T}$ can be computed in $\mathcal{O}(n)$ time using the suffix tree data structure [6,7]. Many different problems based on this notion have already been studied. Pei et al. [4] considered the following problem on the so-called position (or point) queries. Given a position $i$ of $\mathsf{T}$, return a shortest unique substring of $\mathsf{T}$ covering $i$. The authors gave an $\mathcal{O}(n^2)$-time and $\mathcal{O}(n)$-space algorithm, which finds the shortest unique substring covering every position of $\mathsf{T}$. Since then, the problem has

been revisited and optimal $\mathcal{O}(n)$-time algorithms have been presented by Ileri et al. [8] and Tsuruta et al. [9]. Several other variants of this problem have been investigated [10–19].

We introduce a natural generalization of the shortest unique substring problem. Specifically, our focus is on the range version of the problem, which we call the Range Shortest Unique Substring (rSUS) problem. The task is to construct a data structure over $\mathsf{T}$ to be able to answer the following type of online queries efficiently. Given a range $[\alpha, \beta]$, return a shortest substring $\mathsf{T}[k, k+h-1]$ of $\mathsf{T}$ with exactly one occurrence (starting position) in $[\alpha, \beta]$; i.e., $k \in [\alpha, \beta]$, there is no $k' \in [\alpha, \beta]$ $(k' \neq k)$, such that $\mathsf{T}[k, k+h-1] = \mathsf{T}[k', k'+h-1]$, and $h$ is minimal. Note that this substring, $\mathsf{T}[k, k+h-1]$, may end at a position $k+h-1 > \beta$. Further note that there may be multiple shortest unique substrings.

Range queries are a classic data structure topic [20–22]. A range query $q = f(A, i, j)$ on an array of $n$ elements over some set $S$, denoted by $A[1, n]$, takes two indices $1 \leq i \leq j \leq n$, a function $f$ defined over arrays of elements of $S$, and outputs $f(A[i, j]) = f(A[i], \ldots, A[j])$. Range query data structures have also been specifically considered for strings [23–26]. For instance, in bioinformatics applications we are often interested in finding regularities in certain regions of a DNA sequence [27–31]. In the Range–LCP problem, defined by Amir et al. [23], the task is to construct a data structure over $\mathsf{T}$ to be able to answer the following type of online queries efficiently. Given a range $[\alpha, \beta]$, return $i, j \in [\alpha, \beta]$ such that the length of the longest common prefix of $\mathsf{T}[i, n]$ and $\mathsf{T}[j, n]$ is maximal among all pairs of suffixes within this range. The state of the art is an $\mathcal{O}(n)$-word data structure supporting $\mathcal{O}(\log^{\mathcal{O}(1)} n)$-time (polylogarithmic-time) queries [25] (see also [26,32]).

### *1.1. Main Problem and Main Results*

An alphabet $\Sigma$ is a finite nonempty set of elements called letters. We fix a string $\mathsf{T}[1, n] = \mathsf{T}[1] \cdots \mathsf{T}[n]$ over $\Sigma$. The length of $\mathsf{T}$ is denoted by $|\mathsf{T}| = n$. By $\mathsf{T}[i, j] = \mathsf{T}[i] \cdots \mathsf{T}[j]$, we denote the substring of $\mathsf{T}$ starting at position $i$ and ending at position $j$ of $\mathsf{T}$. We say that another string $\mathsf{P}$ has an occurrence in $\mathsf{T}$ or, more simply, that $\mathsf{P}$ occurs in $\mathsf{T}$ if $\mathsf{P} = \mathsf{T}[i, i+|\mathsf{P}|-1]$, for some $i$. Thus, we characterize an occurrence of $\mathsf{P}$ by its starting position $i$ in $\mathsf{T}$. A prefix of $\mathsf{T}$ is a substring of $\mathsf{T}$ of the form $\mathsf{T}[1, i]$ and a suffix of $\mathsf{T}$ is a substring of $\mathsf{T}$ of the form $\mathsf{T}[i, n]$.

We next formally define the main problem considered in this paper.

**Problem** rSUS
*Preprocess:* String $\mathsf{T}[1, n]$.
*Query:* Range $[\alpha, \beta]$, where $1 \leq \alpha \leq \beta \leq n$.
*Output:* $(p, \ell)$ such that $\mathsf{T}[p, p+\ell-1]$ is a shortest string with exactly one occurrence in $[\alpha, \beta]$.

If $\alpha = \beta$ the answer $(\alpha, 1)$ is trivial. So, in the rest we assume that $\alpha < \beta$.

**Example 1.** *Given* $\mathsf{T} = \underset{1\,2\,3\,4\,5\,6\,7\,8\,9\,10\,11\,12\,13\,14\,15\,16\,17\,18\,19\,20\,21}{\boldsymbol{caabcaddaacaddaaaabac}}$ *and a query* $[\alpha, \beta] = [5, 16]$, *we need to find a shortest substring of* $\mathsf{T}$ *with exactly one occurrence in* $[5, 16]$. *The output here is* $(p, \ell) = (10, 2)$, *because* $\mathsf{T}[10, 11] = \boldsymbol{ac}$ *is the shortest substring of* $\mathsf{T}$ *with exactly one occurrence in* $[5, 16]$.

Our main results are summarized below. We consider the standard word-RAM model of computations with $w$-bit machine words, where $w = \Omega(\log n)$, for stating our results.

**Theorem 1.** *We can construct an* $\mathcal{O}(n \log n)$*-word data structure that can answer any* rSUS *query on* $T[1, n]$ *in* $\mathcal{O}(\log_w n)$ *time.*

**Theorem 2.** *We can construct an* $\mathcal{O}(n)$*-word data structure that can answer any* rSUS *query on* $T[1, n]$ *in* $\mathcal{O}(\sqrt{n} \log^{\epsilon} n)$ *time, where* $\epsilon > 0$ *is an arbitrarily small constant.*

### *1.2. Paper Organization*

In Section 2, we prove Theorem 1 and, in Section 3, we prove Theorem 2. We conclude this paper in Section 4 with some future proposals. An early version of this paper appeared as [33]. When compared to that early version ([33]), Theorem 2 is new.

## 2. An $\mathcal{O}(n \log n)$-Word Data Structure

Our construction is based on ingredients, such as the suffix tree [7], heavy-light decomposition [34], and a geometric data structure for rectangle stabbing [35]. Let us start with some definitions.

**Definition 1.** *For a position $k \in [1, n]$ and $h \geq 1$, we define* $\mathsf{Prev}(k, h)$ *and* $\mathsf{Next}(k, h)$*, as follows:*

$$\mathsf{Prev}(k, h) = \max_j \{\{j < k \mid \mathsf{T}[k, k+h-1] = \mathsf{T}[j, j+h-1]\} \cup \{-\infty\}\}$$
$$\mathsf{Next}(k, h) = \min_j \{\{j > k \mid \mathsf{T}[k, k+h-1] = \mathsf{T}[j, j+h-1]\} \cup \{+\infty\}\}.$$

Intuitively, let $x$ and $y$ be the occurrences of $\mathsf{T}[k, k+h-1]$ right before and right after the position $k$, respectively. Subsequently, $\mathsf{Prev}(k, h) = x$ and $\mathsf{Next}(k, h) = y$. If $x$ (resp., $y$) does not exist, then $\mathsf{Prev}(k, h) = -\infty$ (resp., $\mathsf{Next}(k, h) = +\infty$).

**Definition 2.** *Let $k \in [a, b]$. We define $\lambda(a, b, k)$, as follows:*

$$\lambda(a, b, k) = \min\{h \mid \mathsf{Prev}(k, h) < a \ \textbf{\textit{and}} \ \mathsf{Next}(k, h) > b\}.$$

Intuitively, $\lambda(a, b, k)$ denotes the length of the shortest substring that starts at position $k$ with exactly one occurrence in $[a, b]$.

**Definition 3.** *For a position $k \in [1, n]$, we define $C_k$, as follows:*

$$C_k = \{h > 1 \mid (\mathsf{Next}(k, h), \mathsf{Prev}(k, h)) \neq (\mathsf{Next}(k, h-1), \mathsf{Prev}(k, h-1))\} \cup \{1\}$$

**Example 2.** *(Running Example for Definition 3) Let* $\mathsf{T} = \underset{1\,2\,3\,4\,5\,6\,7\,8\,9\,10\,11\,12\,13\,14\,15\,16\,17\,18\,19\,20\,21}{\textbf{\textit{caabcaddaacaddaaaabac}}}$ *and* $k = 10$*. We have that* $(\mathsf{Next}(10, 1), \mathsf{Prev}(10, 1)) = (12, 9)$, $(\mathsf{Next}(10, 2), \mathsf{Prev}(10, 2)) = (20, -\infty)$, *and* $(\mathsf{Next}(10, 3), \mathsf{Prev}(10, 3)) = (+\infty, -\infty)$*. Thus,* $C_{10} = \{2, 3\} \cup \{1\} = \{1, 2, 3\}$.

Intuitively, $C_k$ stores the set of candidate lengths for the shortest unique substrings starting at position $k$. We make the following observation.

**Observation 1.** *$\lambda(a, b, k) \in C_k$, for any $1 \leq a \leq b \leq n$.*

**Example 3.** *(Running Example for Observation 1) Let* $\mathsf{T} = \underset{1\,2\,3\,4\,5\,6\,7\,8\,9\,10\,11\,12\,13\,14\,15\,16\,17\,18\,19\,20\,21}{\textbf{\textit{caabcaddaacaddaaaabac}}}$ *and $k = 10$. We have that $C_{10} = \{1, 2, 3\}$. For $a = 5$ and $b = 16$, $\lambda(5, 16, 10) = 2$, denoting substring* ***ac****. For $a = 5$ and $b = 20$, $\lambda(5, 20, 10) = 3$, denoting substring* ***aca****.*

The following combinatorial lemma is crucial for efficiency.

**Lemma 1.** *$\sum_k |C_k| = \mathcal{O}(n \log n)$.*

**Proof.** The proof of Lemma 1 is deferred to Section 2.1. □

We are now ready to present our construction. By Observation 1, for a given query range $[\alpha, \beta]$, the answer $(p, \ell)$ that we are looking for is the pair $(k, h)$ with the minimum $h$ under the following conditions: $k \in [\alpha, \beta]$, $h \in C_k$, $\mathsf{Prev}(k, h) < \alpha$ and $\mathsf{Next}(k, h) > \beta$. Equivalently, $(p, \ell)$ is the pair $(k, h)$

with the minimum $h$, such that $h \in C_k$, $\alpha \in (\mathsf{Prev}(k,h),k]$, and $\beta \in [k,\mathsf{Next}(k,h))$. We map each $h \in C_k$ into a weighted rectangle $R_{k,h}$ with weight $h$, which is defined as follows:

$$R_{k,h} = [\mathsf{Prev}(k,h)+1,k] \times [k,\mathsf{Next}(k,h)-1].$$

Let $\mathcal{R}$ be the set of all such rectangles, then the lowest weighted rectangle in $\mathcal{R}$ stabbed by the point $(\alpha,\beta)$ is $R_{p,\ell}$. In short, an rSUS query on $\mathsf{T}[1,n]$ with an input range $[\alpha,\beta]$ can be reduced to an equivalent top-1 rectangle stabbing query on a set $\mathcal{R}$ of rectangles with input point $(\alpha,\beta)$. In the 2-d Top-1 Rectangle Stabbing problem, we preprocess a set of weighted rectangles in 2-d, so that given a query point $q$ the task is to report the largest (or lowest) weighted rectangles containing $q$ [35]. Similarly, here, the task is to report the lowest weighted rectangle in $\mathcal{R}$ containing the point $(\alpha,\beta)$ (see Figure 1 for an illustration). By Lemma 1, we have that $|\mathcal{R}| = \mathcal{O}(n \log n)$. Therefore, by employing the optimal data structure for top-1 rectangle stabbing presented by Chan et al. [35], which takes $\mathcal{O}(|\mathcal{R}|)$-word space supporting $\mathcal{O}(\log_w |\mathcal{R}|)$-time queries, we arrive at the space-time trade-off of Theorem 1. This completes our construction.

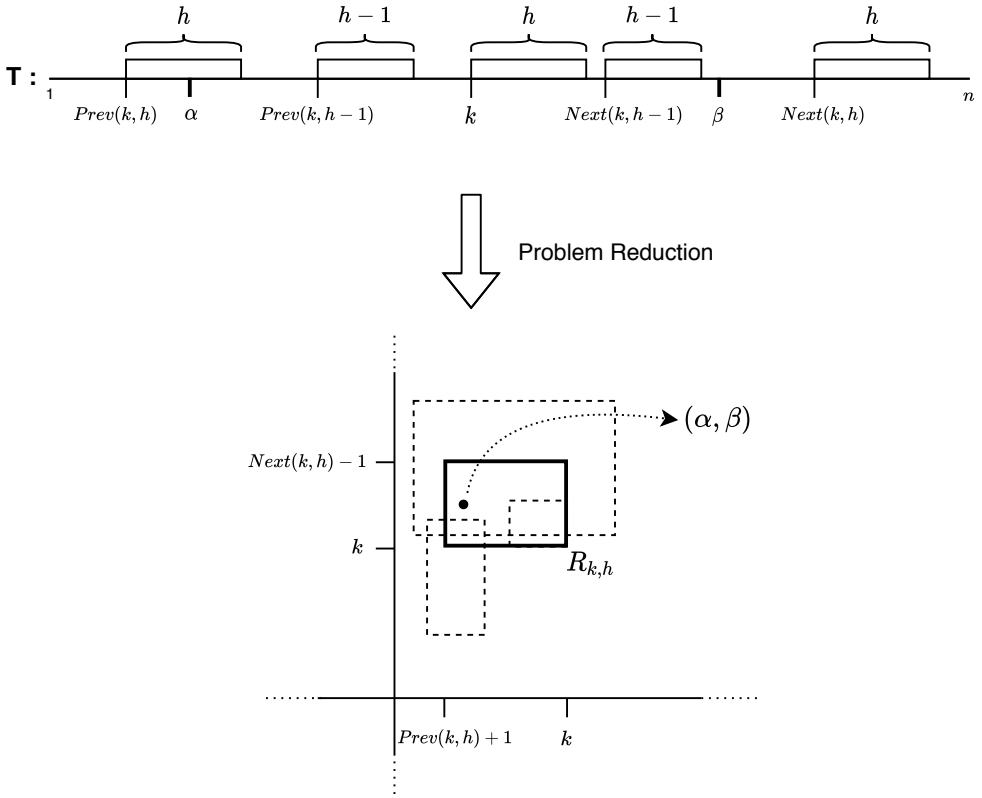


**Figure 1.** Illustration of the problem reduction: $(k,h)$ is the output of the rSUS problem with query range $[\alpha,\beta]$, where $h = \lambda(\alpha,\beta,k) \in C_k$. $R_{k,h}$ is the lowest weighted rectangle in $\mathcal{R}$ containing the point $(\alpha,\beta)$.

### 2.1. Proof of Lemma 1

Let $\mathsf{lcp}(i,j)$ denote the length of the longest common prefix of the suffixes of $\mathsf{T}$ starting at positions $i$ and $j$ in $\mathsf{T}$. Additionally, let $S$ denote the set of all $(x,y)$ pairs, such that $1 \leq x < y \leq n$ and $\mathsf{lcp}(x,y) > \mathsf{lcp}(x,z)$, for all $z \in [x+1,y-1]$. The proof can be broken down into Lemma 2 and Lemma 3.

**Lemma 2.** $\sum_k |C_k| = \mathcal{O}(|S|)$.

**Proof.** Let us fix a position $k$. Let

$$C'_k = \{h > 1 \mid \mathsf{Prev}(k,h) \neq \mathsf{Prev}(k,h-1)\}$$

$$C''_k = \{h > 1 \mid \mathsf{Next}(k,h) \neq \mathsf{Next}(k,h-1)\}.$$

Clearly we have that $C_k = C'_k \cup C''_k \cup \{1\}$.

The following statements can be deduced by a simple contradiction argument:

1. Let $i = \mathsf{Prev}(k,h) \neq -\infty$, where $h \in C'_k$, then $i = \mathsf{Prev}(k, \mathsf{lcp}(i,k))$
2. Let $j = \mathsf{Next}(k,h) \neq \infty$, where $h \in C''_k$, then $j = \mathsf{Next}(k, \mathsf{lcp}(k,j))$.

Figure 2 illustrates the proof for the first statement. The second one can be proved in a similar fashion.

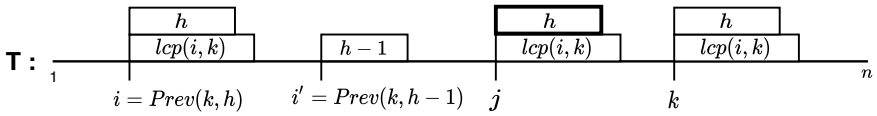


**Figure 2.** Let $h \in C'_k$ and $i = \mathsf{Prev}(k,h)$. By contradiction, assume that there exists $j \in (i,k)$ such that $j = \mathsf{Prev}(k, \mathsf{lcp}(i,k))$. Since $h \leq \mathsf{lcp}(i,k)$, $\mathsf{T}[j, j+h-1] = \mathsf{T}[k, k+h-1]$. This is a contradiction with $i = \mathsf{Prev}(k,h)$. Thus, $i = \mathsf{Prev}(k, \mathsf{lcp}(i,k))$.

Clearly, $|C'_k|$ is proportional to the number of $(i,k)$ pairs, such that $\mathsf{lcp}(i,k) \neq 0$ and $i = \mathsf{Prev}(k, \mathsf{lcp}(i,k))$. Similarly, $|C''_k|$ is proportional to the number of $(k,j)$ pairs, such that $\mathsf{lcp}(k,j) \neq 0$ and $j = \mathsf{Next}(k, \mathsf{lcp}(k,j))$. Therefore, $\sum_k |C_k|$ is proportional to the number of $(x,y)$ pairs, such that $\mathsf{lcp}(x,y) \neq 0$ and $\mathsf{lcp}(x,y) > \mathsf{lcp}(x,z)$, for all $z \in [x+1, y-1]$. This completes the proof of Lemma 2. □

**Lemma 3.** $|S| = \mathcal{O}(n \log n)$.

**Proof.** Consider the suffix tree data structure of string $\mathsf{T}[1,n]$, which is a compact trie of the $n$ suffixes of $\mathsf{T}$ appended with a letter $\$ \notin \Sigma$ [7]. This suffix tree consists of $n+1$ leaves (one for each suffix of $\mathsf{T}$) and at most $n$ internal nodes. The edges are labeled with substrings of $\mathsf{T}$. Let $u$ be the lowest common ancestor of the leaves corresponding to the strings $\mathsf{T}[x,n]\$$ and $\mathsf{T}[y,n]\$$. Subsequently, the concatenation of the edge labels on the path from the root to $u$ is exactly the longest common prefix of $\mathsf{T}[x,n]\$$ and $\mathsf{T}[y,n]\$$. For any node $u$, we denote, by $\mathsf{size}(u)$, the total number of leaf nodes of the subtree rooted at $u$.

We decompose the nodes in the suffix tree into light and heavy nodes. The root node is light and for any internal node, exactly one child is heavy. Specifically, the heavy child is the one having the largest number of leaves in its subtree (ties are broken arbitrarily). All other children are light. This tree decomposition is known as heavy-light decomposition. We have the following critical observation. Any path from the root to a leaf node contains many nodes; however, the number of light nodes is at most $\log n$ [34,36]. Additionally, corresponding to the $n+1$ leaves of the suffix tree, there are $n+1$ paths from the root to the leaves. Therefore, the sum of subtree sizes over all light nodes is $\mathcal{O}(n \log n)$.

We are now ready to complete the proof. Let $S_u \subseteq S$ denote the set of pairs $(x,y)$, such that the lowest common ancestor of the leaves corresponding to suffixes $\mathsf{T}[x,n]\$$ and $\mathsf{T}[y,n]\$$ is $u$. Clearly, the paths from the root to the leaves that correspond to suffixes $\mathsf{T}[x,n]\$$ and $\mathsf{T}[y,n]\$$ pass from two distinct children of node $u$ and then at least one of the two must be a light node. There are two cases. In the first case, both leaves are under the light children. In the second case, one leaf is under a light child and the other is under the heavy child. In both cases, we have at least one leaf under a light node. If we fix the leaf that is under the light node, we can enumerate the total number of pairs based on the subtree size of the light nodes. Therefore, $|S_u|$ is at most twice the sum of $\mathsf{size}(\cdot)$ over all light

children of $u$. Since $|S| = \sum_u |S_u|$, we can bound $|S|$ by the sum of $\mathsf{size}(\cdot)$ over all light nodes in the suffix tree, which is $\mathcal{O}(n \log n)$. This completes the proof of Lemma 3. □

## 3. An $\mathcal{O}(n)$-Word Data Structure

This section is dedicated to proving Theorem 2. For simplicity, we only focus on the computation of the length $\ell$ of the output $(p, \ell)$.

Let $\mathsf{SA}$ be the suffix array of string $\mathsf{T}$ of length $n$ , which is a permutation of $\{1, \cdots, n\}$, such that $\mathsf{SA}[i] = j$ if $\mathsf{T}[j, n]$ is the $i$th lexicographically smallest suffix of $\mathsf{T}$ [37]. Further, let $\mathsf{SA}^{-1}$ be the *inverse suffix array* of string $\mathsf{T}$ of length $n$, which is a permutation of $\{1, \cdots, n\}$, such that $\mathsf{SA}^{-1}[\mathsf{SA}[i]] = i$. Moreover, $\mathsf{SA}$ of $\mathsf{T}$ can be constructed in linear time and space [38,39].

We observe that an $\mathcal{O}(\beta - \alpha + 1)$-time solution is straightforward with the aid of the suffix tree of $\mathsf{T}$ as follows. First, identify those leaves corresponding to the suffixes starting within $[\alpha, \beta]$ using the inverse suffix array of $\mathsf{T}$ and mark them. Subsequently, for each marked leaf, identify its lowest ancestor node (and double mark it), such that a marked neighbor is also under it. This can be done via at most two $\mathcal{O}(1)$-time Lowest Common Ancestor (LCA) queries over the suffix tree of $\mathsf{T}$ while using $\mathcal{O}(n)$ additional space [22]. Afterwards, find the minimum over the string-depth of all double-marked nodes, add 1 to it, and report it as the length $\ell$. The correctness is readily verified.

We employ the above procedure when $\beta - \alpha + 1 < 3\Delta$, where $\Delta$ is a parameter to be set later. We now consider the case when $\beta - \alpha + 1 \geq 3\Delta$. Note that $\ell$ is the smallest element in $S^* = \{\lambda(\alpha, \beta, k) \mid k \in [\alpha, \beta]\}$. Let $\alpha'$ be the smallest number after $\alpha$ and $\beta'$ be the largest number before $\beta$, such that $\alpha'$ and $\beta'$ are multiples of $\Delta$. Subsequently, $S^*$ can be written as the union of $S' = \{\lambda(\alpha, \beta, k) \mid k \in [\alpha, \alpha' - 1] \cup [\beta' + 1, \beta]\}$ and $S'' = \{\lambda(\alpha, \beta, k) \mid k \in [\alpha', \beta']\}$. Furthermore, $S''$ can be written as $S''_1 \cup S''_2 \cup S''_3$, where

- $S''_1 = \{h = \lambda(\alpha, \beta, k) \mid k \in [\alpha', \beta'], \mathsf{Prev}(k, h) \in [\alpha' - \Delta, \alpha - 1]\}$
- $S''_2 = \{h = \lambda(\alpha, \beta, k) \mid k \in [\alpha', \beta'], \mathsf{Next}(k, h) \in [\beta + 1, \beta' + \Delta]\}$
- $S''_3 = \{h = \lambda(\alpha, \beta, k) \mid k \in [\alpha', \beta'], \mathsf{Prev}(k, h) \in (-\infty, \alpha' - \Delta - 1], \mathsf{Next}(k, h) \in [\beta' + \Delta + 1, \infty)\}$.

Our construction is based on a solution to the Orthogonal Range Predecessor/Successor in 2-d problem. A set $\mathcal{P}$ of $n$ points in an $[1, n] \times [1, n]$ grid can be preprocessed into a linear-space data structure, such that the following queries can be answered in $\mathcal{O}(\log^\epsilon n)$ time per query [40]:

- $\mathsf{ORQ}([x', x''], [-\infty, y'']) = \arg\max_j \{(i, j) \in \mathcal{P} \cap [x', x''] \times [-\infty, y'']\}$
- $\mathsf{ORQ}([-\infty, x''], [y', y'']) = \arg\max_i \{(i, j) \in \mathcal{P} \cap [-\infty, x''] \times [y', y'']\}$
- $\mathsf{ORQ}([x', x''], [y', +\infty]) = \arg\min_j \{(i, j) \in \mathcal{P} \cap [x', x''] \times [y', +\infty]\}$
- $\mathsf{ORQ}([x', +\infty], [y', y'']) = \arg\min_i \{(i, j) \in \mathcal{P} \cap [x', +\infty] \times [y', y'']\}$.

We next show how to maintain additional structures, so that the smallest element in each of the above sets can be efficiently computed and, thus, the smallest among them can be reported as $\ell$.

- Computing the Smallest Element in $S'$: for each $k \in [\alpha, \alpha' - 1] \cup [\beta' + 1, \beta]$, we compute $\lambda(\alpha, \beta, k)$ and report the smallest among them. We handle each $\lambda(\alpha, \beta, k)$ query in time $\mathcal{O}(\log^\epsilon n)$, as follows: first find the leaf corresponding to the string position $k$ in the suffix tree of $\mathsf{T}$, then the last (resp., first) leaf on its left (resp., right) side, such that the string position $x$ (resp., $y$) corresponding to it is in $[\alpha, \beta]$, and report $1 + \max\{\mathsf{lcp}(k, x), \mathsf{lcp}(k, y)\}$. To efficiently enable the computation of $x$ (resp., $y$), we preprocess the suffix array into an $\mathcal{O}(n)$-word data structure that can answer orthogonal range predecessor (resp., successor) queries in $\mathcal{O}(\log^\epsilon n)$ time [40].
- Computing the Smallest Element in $S''_1$: for each $r \in [\alpha' - \Delta, \alpha - 1]$, we compute the smallest element in $\{h = \lambda(\alpha, \beta, k) \mid k \in [\alpha', \beta'], \mathsf{Prev}(k, h) = r\}$ and report the smallest among them. The procedure is the following: find the leaf corresponding to the string position $r$ in the suffix tree of $\mathsf{T}$ and the last (resp., first) leaf on its left (resp., right) side, such that its corresponding string position $x$ (resp., $y$) is in $[\alpha', \beta']$ (via orthogonal range successor/predecessor queries as

earlier). Subsequently, $t = \max\{\mathsf{lcp}(r, x), \mathsf{lcp}(r, y)\}$ is the length of the longest prefix of $\mathsf{T}[r, n]$ with an occurrence $d$ in $[\alpha', \beta']$. However, we need to verify whether occurrence $d$ is unique and its $\mathsf{Prev}(d, t) = r$. For this, find the two leftmost occurrences of $\mathsf{T}[r, r + t - 1]$ after $r$, denoted by $x'$ and $y'$ ($x' < y'$), via two orthogonal range successor queries. If $y'$ does not exist, set $y' = +\infty$. Then report $\lambda(\alpha, \beta, d)$ if $\alpha' \leq x' \leq \beta' < y'$. Otherwise, report $+\infty$.

- Computing the Smallest Element in $S_2''$: for each $r \in [\beta + 1, \beta' + \Delta]$, we compute the smallest element in $\{h = \lambda(\alpha, \beta, k) \mid k \in [\alpha', \beta'], \mathsf{Next}(k, h) = r\}$ and report the smallest among them. The procedure is analogous to that of $S_1''$; i.e., find the length $t$ of the longest prefix of $\mathsf{T}[r, n]$ with an occurrence $d$ in $[\alpha', \beta']$. Then, find the two rightmost occurrences of $\mathsf{T}[r, r + t - 1]$ before $r$, denoted by $x'$ and $y'$ ($x' < y'$), via two orthogonal range successor queries. If $x'$ does not exist, set $x' = +\infty$. Subsequently, report $\lambda(\alpha, \beta, d)$ if $x' < \alpha' \leq y' \leq \beta'$. Otherwise, report $+\infty$.
- Computing the Smallest Element in $S_3''$: the set $S_3''$ can be written as $\{\lambda(\alpha' - \Delta, \beta' + \Delta, k) \mid k \in [\alpha', \beta']\}$, which is now dependent only on $\alpha', \beta'$ and $\Delta$. Therefore, our idea is to pre-compute and explicitly store the minimum element in $\{\lambda(a - \Delta, b + \Delta, k) \mid k \in [a, b]\}$ for all $(a, b)$ pairs, where both $a$ and $b$ are multiples of $\Delta$, and for that the desired answer can be retrieved in constant time. The additional space needed is $\mathcal{O}((n/\Delta)^2)$.

We set $\Delta = \lfloor \sqrt{n} \rfloor$. The total space is then $\mathcal{O}(n)$ and total time is $\mathcal{O}(\Delta \log^{\epsilon} n) = \mathcal{O}(\sqrt{n} \log^{\epsilon} n)$. Therefore, we arrive at Theorem 2.

## 4. Final Remarks

We introduced the Range Shortest Unique Substring (rSUS) problem, the range variant of the Shortest Unique Substring problem. We presented a $\mathcal{O}(n \log n)$-word data structure with $\mathcal{O}(\log_w n)$ query time, where $w = \Omega(\log n)$ is the word size, for this problem. We also presented a $\mathcal{O}(n)$-word data structure with $\mathcal{O}(\sqrt{n} \log^{\epsilon} n)$ query time, where $\epsilon > 0$ is an arbitrarily small constant.

We leave the following related questions unanswered:

1. Can we design an $\mathcal{O}(n)$-word data structure for the rSUS problem with polylogarithmic query time?
2. Can we design an efficient solution for the $k$ mismatches/edits variation of the rSUS problem, perhaps using the framework of [41]?
3. Can our reduction from Section 2 be extended to other types of string regularities, such as shortest absent words [42]?

**Author Contributions:** The initial draft of this paper was written by P.A. and S.V.T. S.P.P. wrote the introduction section and modified the whole draft. A.G. reviewed and edited the paper. All authors contributed to the manuscript equally by providing critical feedback and helping with the presentation. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research was supported in part by the U.S. National Science Foundation (NSF) under CCF-1703489.

**Conflicts of Interest:** The authors declare no conflict of interest.

## References

1. Lothaire, M. *Applied Combinatorics on Words*; Cambridge University Press: Cambridge, UK, 2005.
2. Schleiermacher, C.; Ohlebusch, E.; Stoye, J.; Choudhuri, J.V.; Giegerich, R.; Kurtz, S. REPuter: The manifold applications of repeat analysis on a genomic scale. *Nucleic Acids Res.* **2001**, *29*, 4633–4642. [CrossRef]
3. Haubold, B.; Pierstorff, N.; Möller, F.; Wiehe, T. Genome comparison without alignment using shortest unique substrings. *BMC Bioinform.* **2005**, *6*, 123. [CrossRef]
4. Pei, J.; Wu, W.C.; Yeh, M. On shortest unique substring queries. In Proceedings of the 29th IEEE International Conference on Data Engineering (ICDE 2013), Brisbane, Australia, 8–12 April 2013; pp. 937–948. [CrossRef]
5. Khmelev, D.V.; Teahan, W.J. A Repetition Based Measure for Verification of Text Collections and for Text Categorization. In Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Informaion Retrieval, Toronto, ON, Canada, 28 July–1 August 2003; pp. 104–110. [CrossRef]

6. Gusfield, D. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*; Cambridge University Press: Cambridge, UK, 1997. [CrossRef]
7. Weiner, P. Linear Pattern Matching Algorithms. In Proceedings of the 14th Annual Symposium on Switching and Automata Theory (SWAT 1973), Iowa City, IA, USA, 15–17 October 1973; pp. 1–11. [CrossRef]
8. Ileri, A.M.; Külekci, M.O.; Xu, B. Shortest Unique Substring Query Revisited. In Proceedings of the Combinatorial Pattern Matching—25th Annual Symposium (CPM 2014), Moscow, Russia, 16–18 June 2014; pp. 172–181. [CrossRef]
9. Tsuruta, K.; Inenaga, S.; Bannai, H.; Takeda, M. Shortest Unique Substrings Queries in Optimal Time. In Proceedings of the 40th International Conference on Current Trends in Theory and Practice of Computer Science, Nový Smokovec, Slovakia, 26–29 January 2014; pp. 503–513. [CrossRef]
10. Abedin, P.; Külekci, M.O.; V Thankachan, S. A Survey on Shortest Unique Substring Queries. *Algorithms* **2020**, *13*, 224. [CrossRef]
11. Allen, D.R.; Thankachan, S.V.; Xu, B. A Practical and Efficient Algorithm for the k-mismatch Shortest Unique Substring Finding Problem. In Proceedings of the 2018 ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics (BCB 2018), Washington, DC, USA, 29 August–1 September 2018; pp. 428–437. [CrossRef]
12. Ganguly, A.; Hon, W.; Shah, R.; Thankachan, S.V. Space-Time Trade-Offs for the Shortest Unique Substring Problem. In Proceedings of the 27th International Symposium on Algorithms and Computation (ISAAC), Sydney, Australia, 12–14 December 2016; pp. 34:1–34:13. [CrossRef]
13. Ganguly, A.; Hon, W.; Shah, R.; Thankachan, S.V. Space-time trade-offs for finding shortest unique substrings and maximal unique matches. *Theor. Comput. Sci.* **2017**, *700*, 75–88. [CrossRef]
14. Inoue, H.; Nakashima, Y.; Mieno, T.; Inenaga, S.; Bannai, H.; Takeda, M. Algorithms and combinatorial properties on shortest unique palindromic substrings. *J. Discret. Algorithms* **2018**, *52*, 122–132. [CrossRef]
15. Hon, W.; Thankachan, S.V.; Xu, B. In-place algorithms for exact and approximate shortest unique substring problems. *Theor. Comput. Sci.* **2017**, *690*, 12–25. [CrossRef]
16. Mieno, T.; Inenaga, S.; Bannai, H.; Takeda, M. Shortest Unique Substring Queries on Run-Length Encoded Strings. In Proceedings of the 41st International Symposium on Mathematical Foundations of Computer Science MFCS, Kraków, Poland, 22–26 August 2016; pp. 69:1–69:11. [CrossRef]
17. Schultz, D.W.; Xu, B. On k-Mismatch Shortest Unique Substring Queries Using GPU. In Proceedings of the 14th International Symposium, Bioinformatics Research and Applications, Beijing, China, 8–11 June 2018, pp. 193–204. [CrossRef]
18. Mieno, T.; Köppl, D.; Nakashima, Y.; Inenaga, S.; Bannai, H.; Takeda, M. Compact Data Structures for Shortest Unique Substring Queries. In Proceedings of the 26th International Symposium, String Processing and Information Retrieval, Segovia, Spain, 7–9 October 2019; pp. 107–123. [CrossRef]
19. Watanabe, K.; Nakashima, Y.; Inenaga, S.; Bannai, H.; Takeda, M. Shortest Unique Palindromic Substring Queries on Run-Length Encoded Strings. In Proceedings of the 30th International Workshop Combinatorial Algorithms, Pisa, Italy, 23–25 July 2019; pp. 430–441. [CrossRef]
20. Yao, A.C. Space-time Tradeoff for Answering Range Queries (Extended Abstract). In Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing (STOC '82), San Francisco, CA, USA, 5–7 May 1982; pp. 128–136. [CrossRef]
21. Berkman, O.; Vishkin, U. Recursive Star-Tree Parallel Data Structure. *SIAM J. Comput.* **1993**, *22*, 221–242. [CrossRef]
22. Bender, M.A.; Farach-Colton, M. The LCA Problem Revisited. In Proceedings of the 4th Latin American Symposium, LATIN 2000: Theoretical Informatics, Punta del Este, Uruguay, 10–14 April 2000; pp. 88–94. [CrossRef]
23. Amir, A.; Apostolico, A.; Landau, G.M.; Levy, A.; Lewenstein, M.; Porat, E. Range LCP. *J. Comput. Syst. Sci.* **2014**, *80*, 1245–1253. [CrossRef]
24. Amir, A.; Lewenstein, M.; Thankachan, S.V. Range LCP Queries Revisited. In Proceedings of the 22nd International Symposium, String Processing and Information Retrieval, London, UK, 1–4 September 2015; pp. 350–361. [CrossRef]
25. Abedin, P.; Ganguly, A.; Hon, W.; Nekrich, Y.; Sadakane, K.; Shah, R.; Thankachan, S.V. A Linear-Space Data Structure for Range-LCP Queries in Poly-Logarithmic Time. In Proceedings of the 24th International Conference, Computing and Combinatorics, Qing Dao, China, 2–4 July 2018; pp. 615–625. [CrossRef]

26. Ganguly, A.; Patil, M.; Shah, R.; Thankachan, S.V. A Linear Space Data Structure for Range LCP Queries. *Fundam. Inform.* **2018**, *163*, 245–251. [CrossRef]
27. Pissis, S.P. MoTeX-II: structured MoTif eXtraction from large-scale datasets. *BMC Bioinform.* **2014**, *15*, 235. [CrossRef] [PubMed]
28. Almirantis, Y.; Charalampopoulos, P.; Gao, J.; Iliopoulos, C.S.; Mohamed, M.; Pissis, S.P.; Polychronopoulos, D. On avoided words, absent words, and their application to biological sequence analysis. *Algorithms Mol. Biol.* **2017**, *12*, 5:1–5:12. [CrossRef]
29. Ayad, L.A.K.; Pissis, S.P.; Polychronopoulos, D. CNEFinder: finding conserved non-coding elements in genomes. *Bioinformatics* **2018**, *34*, i743–i747. [CrossRef]
30. Iliopoulos, C.S.; Mohamed, M.; Pissis, S.P.; Vayani, F. Maximal Motif Discovery in a Sliding Window. In Proceedings of the 25th International Symposium, String Processing and Information Retrieval, Lima, Peru, 9–11 October 2018; pp. 191–205. [CrossRef]
31. Almirantis, Y.; Charalampopoulos, P.; Gao, J.; Iliopoulos, C.S.; Mohamed, M.; Pissis, S.P.; Polychronopoulos, D. On overabundant words and their application to biological sequence analysis. *Theor. Comput. Sci.* **2019**, *792*, 85–95. [CrossRef]
32. Matsuda, K.; Sadakane, K.; Starikovskaya, T.; Tateshita, M. Compressed Orthogonal Search on Suffix Arrays with Applications to Range LCP. In Proceedings of the 31st Annual Symposium on Combinatorial Pattern Matching, Copenhagen, Denmark, 17–19 June 2020; pp. 23:1–23:13. [CrossRef]
33. Abedin, P.; Ganguly, A.; Pissis, S.P.; Thankachan, S.V. Range Shortest Unique Substring Queries. In Proceedings of the 26th International Symposium, String Processing and Information Retrieval, Segovia, Spain, 7–9 October 2019; pp. 258–266. [CrossRef]
34. Sleator, D.D.; Tarjan, R.E. A Data Structure for Dynamic Trees. In Proceedings of the 13th Annual ACM Symposium on Theory of Computing, Milwaukee, WI, USA, 11–13 May 1981; pp. 114–122. [CrossRef]
35. Chan, T.M.; Nekrich, Y.; Rahul, S.; Tsakalidis, K. Orthogonal Point Location and Rectangle Stabbing Queries in 3-d. In Proceedings of the 45th International Colloquium on Automata, Languages, and Programming, Prague, Czech Republic, 9–13 July 2018; pp. 31:1–31:14. [CrossRef]
36. Harel, D.; Tarjan, R.E. Fast Algorithms for Finding Nearest Common Ancestors. *SIAM J. Comput.* **1984**, *13*, 338–355. [CrossRef]
37. Manber, U.; Myers, E.W. Suffix Arrays: A New Method for On-Line String Searches. *SIAM J. Comput.* **1993**, *22*, 935–948. [CrossRef]
38. Farach, M. Optimal Suffix Tree Construction with Large Alphabets. In Proceedings of the 38th Annual Symposium on Foundations of Computer Science (FOCS '97), Miami Beach, FL, USA, 19–22 October 1997; pp. 137–143. [CrossRef]
39. Kärkkäinen, J.; Sanders, P.; Burkhardt, S. Linear work suffix array construction. *J. ACM* **2006**, *53*, 918–936. [CrossRef]
40. Nekrich, Y.; Navarro, G. Sorted Range Reporting. In Proceedings of the 13th Scandinavian Symposium and Workshops (SWAT 2012), Helsinki, Finland, 4–6 July 2012; pp. 271–282. [CrossRef]
41. Thankachan, S.V.; Aluru, C.; Chockalingam, S.P.; Aluru, S. Algorithmic Framework for Approximate Matching Under Bounded Edits with Applications to Sequence Analysis. In Proceedings of the 22nd Annual International Conference, Research in Computational Molecular Biology (RECOMB 2018), Paris, France, 21–24 April 2018; pp. 211–224. [CrossRef]
42. Barton, C.; Héliou, A.; Mouchard, L.; Pissis, S.P. Linear-time computation of minimal absent words using suffix array. *BMC Bioinform.* **2014**, *15*, 388. [CrossRef]